DOI:10.1145/1831407.1831427

Within a decade, P2P has proven to be a technology that enables innovative new services and is used by millions of people every day.

BY RODRIGO RODRIGUES AND PETER DRUSCHEL

Peer-to-Peer Systems

PEER-TO-PEER (P2P) COMPUTING has attracted significant interest in recent years, originally sparked by the release of three influential systems in 1999: the Napster music-sharing system, the Freenet anonymous data store, and the SETI@home volunteer-based scientific computing projects. Napster, for instance, allowed its users to download music directly from each other's computers via the Internet. Because the bandwidth-intensive music downloads occurred directly between users' computers, Napster avoided significant operating costs and was able to offer its service to millions of users for free. Though unresolved legal issues ultimately sealed Napster's fate, the idea of cooperative resource sharing among peers found its way into many other applications.

More than a decade later, P2P technology has gone far beyond music sharing, anonymous data storage, or scientific computing; it now enjoys significant research attention and increasingly widespread use in open software communities and industry alike. Scientists, companies, and open-software organizations use BitTorrent to distribute bulk data such as software updates, data sets, and media files to many nodes;[5] commercial P2P software allows enterprises to distribute news and events to their employees and customers;[29] millions of people use Skype to make video and phone calls;[1] and hundreds of TV channels are available using live streaming applications such as PPLive,[17] CoolStreaming,[38] and the BBC's iPlayer.[4]

The term P2P has been defined in different ways, so we should clarify what exactly we mean by a P2P system. For the purposes of this article, a P2P system is a distributed system with the following properties:

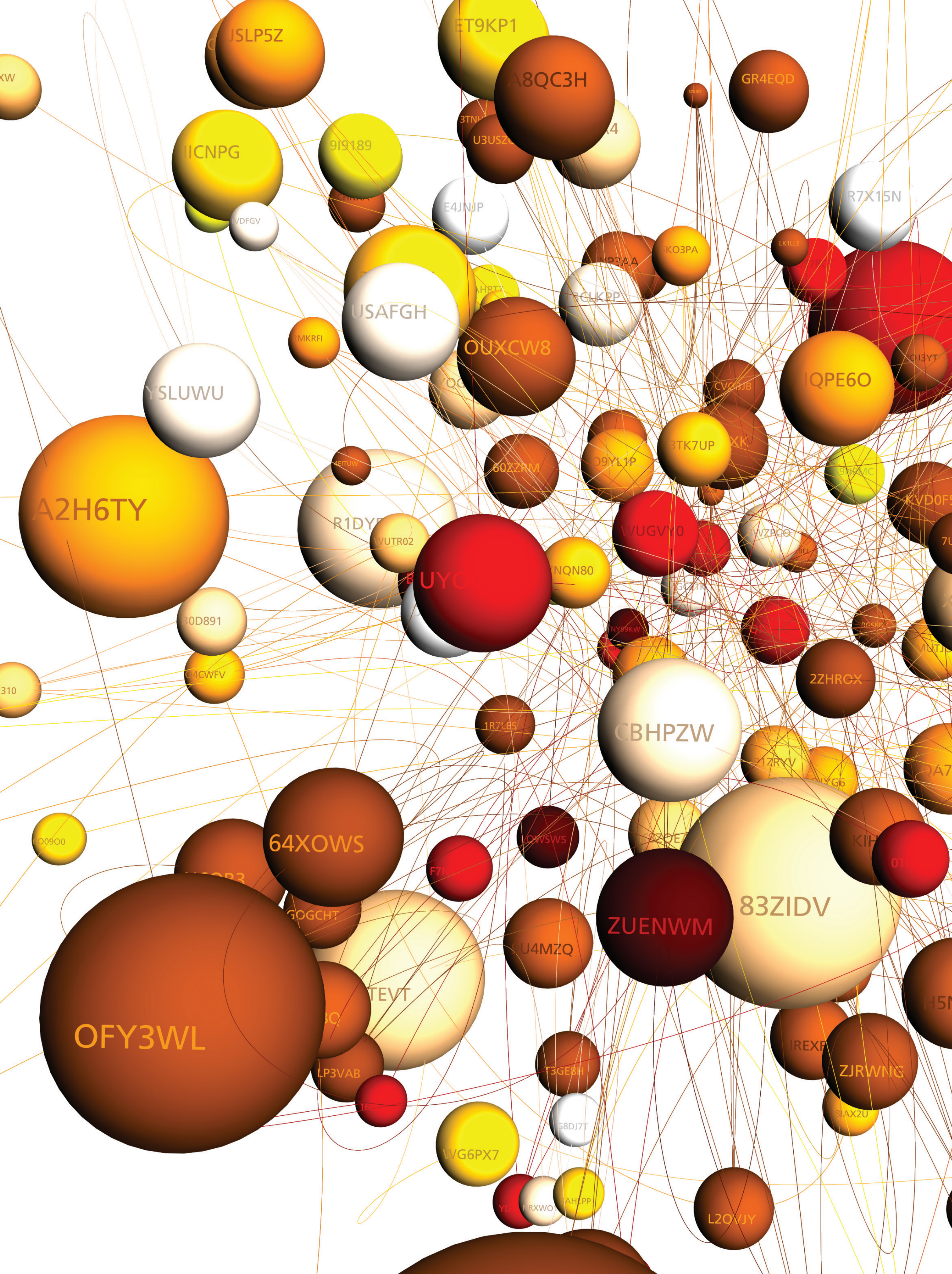
High degree of decentralization. The peers implement both client and server functionality and most of the system's state and tasks are dynamically allocated among the peers. There are few if any dedicated nodes with centralized state. As a result, the bulk of the computation, bandwidth, and storage needed to operate the system are contributed by participating nodes.

Self-organization. Once a node is introduced into the system (typically by providing it with the IP address of a participating node and any necessary

» key insights

- **P2P leverages the computing resources of cooperating users to achieve scalability and organic growth, thus lowering the deployment barrier for innovative new services.**
- **Originally invented for music/data sharing and volunteer computing, P2P systems now enjoy widespread commercial and non-commercial use in content distribution, IPTV, and IP telephony.**
- **The strength of P2P—its independence of dedicated infrastructure and centralized control—is also its weakness, as it presents new technical, commercial, and legal challenges.**
- **P2P technology may turn out to be most valuable as a low-cost deployment vector for experimental, innovative services; those services that prove to be commercially viable can be subsequently combined with centralized, infrastructure-based components.**

ILLUSTRATION BY MARIUS WATZ



JSLP5Z
ET9KP1
A8QC3H
GR4EQD
3TNH
U3USZ
IICNPG
9I9189
E4JNJP
VDFGV
R7X15N
KO3PA
LK1LLE
USAFGH
OUXCW8
MKRFI
YSLUWU
QPE6O
CVG3JB
3TK7UP
O9YL1P
60ZZRM
A2H6TY
R1DY
WUTR02
UYC
NQN80
WUGVY0
KVD0F
0D891
C4CWFV
310
2ZHROX
1R7LE5
CBHPZW
QA7
17RYV
64XOWS
O09OO
LOWSW5
F7N
ZUENWM
83ZIDV
KIH
0T
GOGCHT
OFY3WL
TEVT
U4MZQ
LP3VAB
REXF
ZJRWNG
T3GE8H
G8DJ7T
3JAX2U
WG6PX7
RXWO
AHLPP
L2OVJY
H5N

key material), little or no manual configuration is needed to maintain the system.

Multiple administrative domains. The participating nodes are not owned and controlled by a single organization. In general, each node is owned and operated by an independent individual who voluntarily joins the system.

P2P systems have several distinctive characteristics that make them interesting:

Low barrier to deployment. Because P2P systems require little or no dedicated infrastructure, the upfront investment needed to deploy a P2P service tends to be low when compared to client-server systems.

Organic growth. Because the resources are contributed by participating nodes, a P2P system can grow almost arbitrarily without requiring a "fork-lift upgrade" of existing infrastructure, for example, the replacement of a server with more powerful hardware.

Resilience to faults and attacks. P2P systems tend to be resilient to faults because there are few if any nodes that are critical to the system's operation. To attack or shut down a P2P system, an attacker must target a large proportion of the nodes simultaneously.

Abundance and diversity of resources. Popular P2P systems have an abundance of resources that few organizations would be able to afford individually. The resources tend to be diverse in terms of their hardware and software architecture, network attachment, power supply, geographic location and jurisdiction. This diversity reduces their vulnerability to correlated failure, attack, and even censorship.

As with other technologies (for example, cryptography), the properties of P2P systems lend themselves to desirable and undesirable use. For instance, P2P systems' resilience may help citizens avoid censorship by a totalitarian regime; at the same time, it can be abused to try and hide criminal activity from law enforcement agencies. The scalability of a P2P system can be used to disseminate a critical software update efficiently at a planetary scale, but can also be used to facilitate the illegal distribution of copyrighted content.

Despite having acquired a negative reputation for some of its initial purposes, P2P technologies are increasingly being used for legal applications with enormous business potential, and there is consensus about their ability to lower the barrier for the introduction of innovative technologies. Nevertheless, P2P technology faces many challenges. The decentralized nature of P2P systems raises concerns about manageability, security, and law enforcement. Moreover, P2P applications are affecting the traffic experienced by Internet service providers (ISPs) and threaten to disrupt the current Internet economics. In this article, we briefly sketch important highlights of the technology, its applications, and the challenges it faces.

Applications

Here, we discuss some of the most successful P2P systems and also mention promising P2P systems that have not yet received as much attention.

Sharing and distributing files. Presently, the most popular P2P applications are file sharing (for example, eDonkey) and bulk data distribution (for example, BitTorrent).

Both types of systems can be viewed as successors of Napster. In Napster, users shared a subset of their disk files with other participants, who were able to search for keywords in the file names. Users would then download any of the files in the query results directly from the peer that shared it.

Much of the content shared by Napster users was music, which led to copyright infringement lawsuits. Napster was found guilty and had to shut down its services. Simultaneously, a series of similar P2P systems appeared, most notably Gnutella and FastTrack (better known by one of its client applications, Kazaa). Gnutella, unlike Napster, has no centralized components and is not operated by any single entity (perhaps in part to make it harder to prosecute).

The desire to reduce the download time for very large files lead to the design of BitTorrent,[10] which enables a large set of users to download bulk data quickly and efficiently. The system uses spare upload bandwidth of concurrent downloaders and peers who already have the complete file (either because they are data sources or have finished the download) to assist other downloaders in the system. Unlike file-sharing applications, BitTorrent and other P2P content distribution networks do not include a search component, and users downloading different content are unaware of each other, since they form separate networks. The protocol is widely used for disseminating data, software, or media content.

Streaming media. An increasingly popular P2P application is streaming media distribution and IPTV (delivering digital television service over the Internet). As in file sharing, the idea is to leverage the bandwidth of participating clients to avoid the bandwidth costs of server-based solutions.

Streaming media distribution has stricter timing requirements than downloading bulk data because data must be delivered before the playout deadline to be useful.

Example systems include academic efforts with widespread adoption such as PPLive[17] and CoolStreaming,[38] and commercial products such as BBC's iPlayer[4] and Skinkers LiveStation.[29]

Telephony. Another major use of P2P technology on the Internet is for making audio and video calls, popularized by the Skype application. Skype exploits the resources of participating nodes to provide seamless audiovisual connectivity to its users, regardless of their current location or type of Internet connection. Peers assist those without publicly routable IP addresses to establish connections, thus working around connectivity problems due to firewalls and network address translation, without requiring a centralized infrastructure that handles and forwards calls. Skype reported 520 million registered users at the end of 2009.

Volunteer computing. A fourth important P2P application is volunteer computing. In these systems, users donate their spare CPU cycles to scientific computations, usually in fields such as astrophysics, biology, or climatology. The first system of this type was SETI@home. Volunteers install a screen saver that runs the P2P application when the user is not active. This application downloads blocks containing observational data collected at the Arecibo radio telescope from the SETI@home server. Then the application analyzes this data, searching for possible radio transmissions, and sends the results back to the server.

The success of SETI@home and similar projects led to the development of the BOINC platform,[3] which has been used to develop many cycle-sharing P2P systems in use today. At the time of this writing, BOINC has more than half a million active peers computing on average 5.42 petaFLOPS (floating-point operations per second). For comparison, a modern PC performs on the order of a few tens of GFLOPS (about five orders of magnitude fewer), and the world's fastest supercomputer as of August 2010 has a performance of about 1.76 petaFLOPS.

Other applications. Other types of P2P applications have seen significant use, at least temporarily, but have not reached the same levels of adoption as the systems we describe here. Among them are applications that leverage peer-contributed disk space to provide distributed storage. Freenet[9] aims to combine distributed storage with content distribution, censorship resistance, and anonymity. It is still active, but the properties of the system make it difficult to estimate its actual use. MojoNation[36] was a subsequent project for building a reliable P2P storage system, but it was shut down after proving unable to ensure the availability of data due to unstable membership and other problems.

P2P Web content distribution networks (CDNs) such as CoralCDN[16] and CoDeeN[35] were deployed as research prototypes but gained widespread use. In these systems, a set of cooperating users form a network of Web caches and name servers that replicates Web content as users access it, thereby reducing the load on servers hosting popular content. During its peak usage, CoralCDN received up to 25 million hits per day from one million unique IP addresses.

Many more P2P systems have been designed and prototyped, but either were not deployed publicly or had small deployments. Examples include systems for distributed data monitoring, management and mining,[26,37] massively distributed query processing,[19] cooperative backup,[11] bibliographic databases,[33] serverless email,[24] and archival storage.[23]

Technology developed for P2P applications has also been incorporated into other types of systems. For instance, Dynamo,[13] a storage substrate that Amazon uses internally for many of its services and applications, uses distributed hash tables (DHTs), which we will explain later. Akamai's NetSession[a] client uses P2P downloads to increase performance and reduce the cost of delivering streaming content. Even though these systems are controlled by a single organization and thus do not strictly satisfy our definition of a P2P system, they are based on P2P technology.

While the earliest and most visible P2P systems were mainly file-sharing applications, current uses of P2P technology are much more diverse and include the distribution of data, software, media content, as well as Internet telephony and scientific computing.

While P2P systems are a recent invention, technical predecessors of P2P systems have existed for a long time. Early examples include the NNTP and SMTP news and mail distribution systems, and the Internet routing system. Like P2P systems, these are mostly decentralized systems that rely on resource contributions from their participants. However, the peers in these systems are organizations and the protocols are not self-organizing.

While the earliest and most visible P2P systems were mainly file-sharing applications, current uses of P2P technology are much more diverse and include the distribution of data, software, media content, as well as Internet telephony and scientific computing. Moreover, an increasing number of commercial services and products rely on P2P technology.

How Do P2P Systems Work?

Here, we sketch some of the most important techniques that make P2P systems work. We discuss fundamental architectural choices like the degree of centralization and the structure of the overlay network. As you will see, one of the key challenges is to build an overlay with a routing capability that works well in the presence of a high membership turnover (usually referred to as churn), which is typical of deployed P2P system.[28] We then present solutions to specific problems addressed in the context of P2P systems: application state maintenance, application-level node coordination, and content distribution.

Note that our intention in this presentation is to provide representative

[a] See Akamai NetSession Interface Overview at http://www.akamai.com/html/misc/akamai_client/netsession_interface.html/.

examples of the most interesting techniques rather than try to be exhaustive or precise about a particular system or protocol.

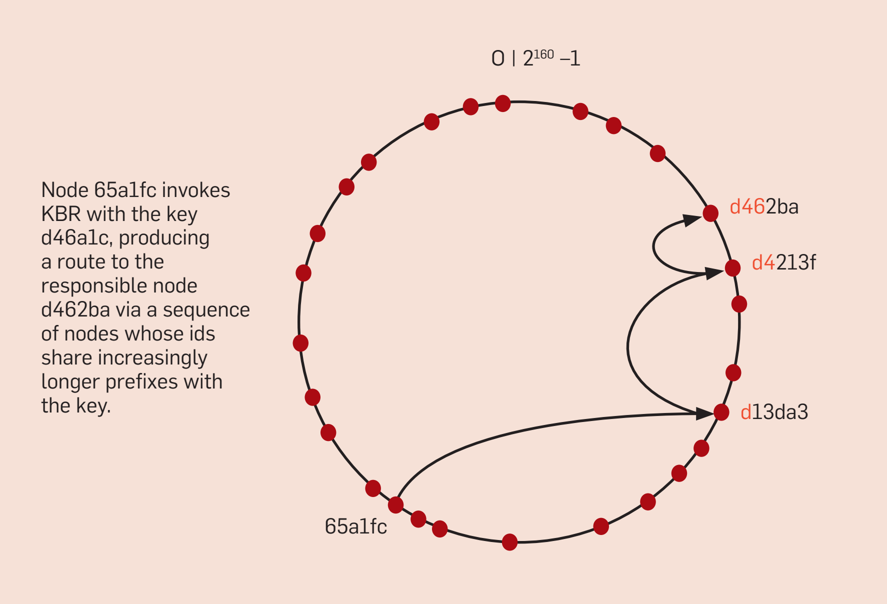
Degree of centralization. We can broadly categorize the architecture of P2P systems according to the presence or absence of centralized components in the system design.

Partly centralized P2P systems have a dedicated controller node that maintains the set of participating nodes and controls the system. For instance, Napster had a Web site that maintained the membership and a content index; early versions of BitTorrent have a "tracker," which is a node that keeps track of the set of nodes uploading and downloading the same content, and periodically provides nodes with a set of peers they can connect to;[10] the BOINC platform for volunteer computing has a site that maintains the membership and assigns compute tasks;[3] and Skype has a central site that provides log-in, account management, and payment.

Resource-intensive operations like transmitting content or computing application functions do not involve the controller. Like general P2P systems, partly centralized P2P systems can provide organic growth and abundant resources. However, they do not necessarily offer the same scalability and resilience because the controller forms a potential bottleneck and a single point of failure and attack. Partly centralized P2P systems are relatively simple and can be managed by a single organization via the controller.

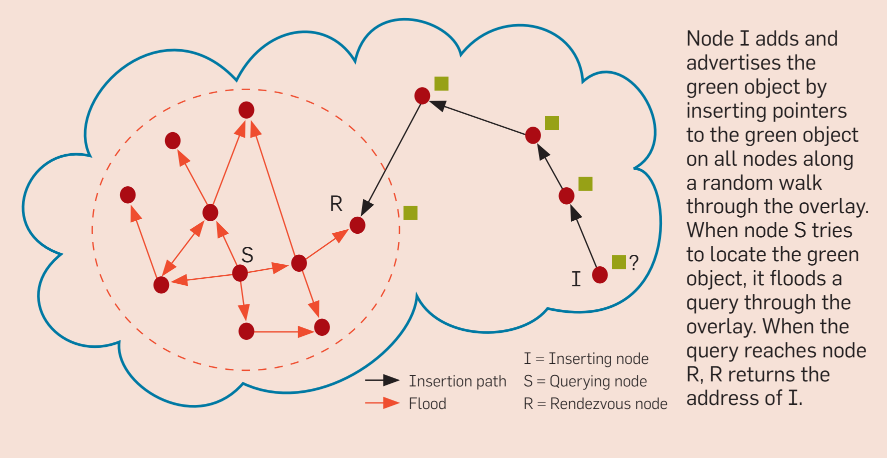
Decentralized P2P system. In a decentralized P2P system, there are no dedicated nodes that are critical for the operation of the system. Decentralized P2P systems have no inherent bottlenecks and can potentially scale very well. Moreover, the lack of dedicated nodes makes them potentially resilient to failure, attack, and legal challenge.

Figure 1. An example KBR implementation.



Node 65a1fc invokes KBR with the key d46a1c, producing a route to the responsible node d462ba via a sequence of nodes whose ids share increasingly longer prefixes with the key.

Figure 2. Locating objects in unstructured overlays.



Node I adds and advertises the green object by inserting pointers to the green object on all nodes along a random walk through the overlay. When node S tries to locate the green object, it floods a query through the overlay. When the query reaches node R, R returns the address of I.

In some decentralized P2P systems, nodes with plenty of resources, high availability and a publicly routable IP address act as supernodes. These supernodes have additional responsibilities, such as acting as a rendez-vous point for nodes behind firewalls, storing state or keeping an index of available content. Supernodes can increase the efficiency of a P2P system, but may also increase its vulnerability to node failure.

Overlay maintenance. P2P systems maintain an overlay network, which can be thought of as a directed graph $G = (N,E)$, where N is the set of participating computers and E is a set of *overlay links*. A pair of nodes connected by a link in E is aware of each other's IP address and communicates directly via the Internet. Here, we discuss how different types of P2P systems maintain their overlay.

In partly centralized P2P systems, new nodes join the overlay by connecting to the controller located at a well-known domain name or IP address (which can be, for instance, hardcoded in the application). Thus, the overlay initially has a star-shaped topology with the controller at the center. Additional overlay links may be formed dynamically among participants that have been introduced by the controller.

In decentralized overlays, newly joining nodes are expected to obtain, through an outside channel, the network address (for example, IP address and port number) of some node that already participates in the system. The address of such a bootstrap node can be obtained, for instance, from a Web site. To join, the new node contacts the bootstrap node.

We distinguish between systems that maintain an unstructured or a structured overlay network.

Unstructured overlays. In an unstructured P2P system, there are no constraints on the links between different nodes, and therefore the overlay graph does not have any particular structure. In a typical unstructured P2P system, a newly joining node forms its initial links by repeatedly performing a random walk through the overlay starting at the bootstrap node and requesting a link to the node where the walk terminates. Nodes acquire additional links (for example, by performing more random walks) whenever their degree falls below the desired minimum; they refuse link requests when their current degree is at its maximum.

The minimum node degree is typically chosen to maintain connectivity in the overlay despite node failures and membership churn. A maximum degree is maintained to bound the overhead associated with maintaining overlay links.

Structured overlays. In a structured overlay, each node has a unique identifier in a large numeric key space, for example, the set of 160-bit integers. Identifiers are chosen in a way that makes them uniformly distributed in that space. The overlay graph has a specific structure; a node's identifier determines its position within that structure and constrains its set of overlay links.

Keys are also used when assigning responsibilities to nodes. The key space is divided among the participating nodes, such that each key is mapped to exactly one of the current overlay nodes via a simple function. For instance, a key may be mapped to the node whose identifier is the key's closest counterclockwise successor in the key space. In this technique the key space is considered to be circular (that is, the id zero succeeds the highest id value) to account for the fact that there may exist keys greater than all node identifiers.

The overlay graph structure is chosen to enable efficient key-based routing. Key-based routing implements the primitive $KBR(n_0, k)$. Given a starting node n_0 and a key k, KBR produces a path, that is, a sequence of overlay nodes that ends in the node responsible for k. As will become clear in subsequent sections, KBR is a powerful primitive.

Many implementations of key-based routing exist.[18,27,32] In general, they strike a balance between the amount of routing state required at each node and the number of forwarding hops required to deliver a message. Typical implementations require an amount of per-node state and a number of forwarding hops that are both logarithmic in the size of the network.

Figure 1 illustrates an example of a key-based routing scheme. Node 65a1fc invokes KBR with the key d46a1c, producing a route via a sequence of nodes whose ids share increasingly longer prefixes with the key. Eventually the message reaches the node with id d462ba, which has sufficient knowledge about its neighboring nodes to determine that it is responsible for the target key. Though not depicted, the reply can be forwarded directly to the invoking node.

Summary. We have seen how the overlay network is formed and maintained in different types of P2P systems. In partly centralized P2P systems, the controller facilitates the overlay formation.

In other P2P systems, overlay maintenance is fully decentralized. Compared to an unstructured overlay network, a structured overlay network invests additional resources to maintain a specific graph structure. In return, structured overlays are able to perform key-based routing efficiently.

The choice between an unstructured and a structured overlay depends on how useful key-based routing is for the application, and also on the frequency of overlay membership events. As we will discuss, key-based routing can reliably and efficiently locate uniquely identified data items and maintain spanning trees among member nodes. However, maintaining a structured overlay in a high-churn environment has an associated cost, which may not be worth paying if the application does not require the functionality provided by key-based routing.

Some P2P systems use both structured and unstructured overlays. A recent ("trackerless") version of BitTorrent, for instance, uses key-based routing to choose tracker nodes, but builds an unstructured overlay to disseminate the content.

Distributed state. Most P2P systems maintain some application-specific distributed state. Without loss of generality, we consider that state as a collection of objects with unique keys. Maintaining this collection of state objects in a distributed manner, that is, providing mechanisms for object placement and locating objects, are key tasks in such systems.

Partly centralized systems. In partly centralized P2P systems, an object is typically stored at the node that inserted the object, as well as any nodes that have subsequently downloaded the object. The controller node maintains information about which objects exist in the system, their keys, names and other attributes, and which nodes are currently storing those objects. Queries for a given key, or a set of keywords that match an object's name or attributes, are directed to the controller, which responds with a set of nodes from which the corresponding object(s) can be downloaded.

Unstructured systems. As in partly centralized systems, content is typically stored at the node that introduced the content to the system, and replicated at other downloaders. To make it easier to find content, some systems place copies of (or pointers to) an inserted object on additional nodes, for instance, along a random walk path through the overlay.

To locate an object, a querying node typically floods a request message through the overlay. The query can specify the desired object by its key, metadata, or keywords. A node that receives a query and has a matching object (or a pointer to a matching object), responds to the querying node. Figure 2 illustrates this process. In this case, node I inserts an object into the system and holds its only copy, but inserts pointers to the object on all nodes along a random walk that ends in node R. When node S tries to locate the object, it floods a query, first, to all nodes that are at a distance of one hop, then to all nodes two hops away. In the last step the query reaches node R, which returns the address of I.

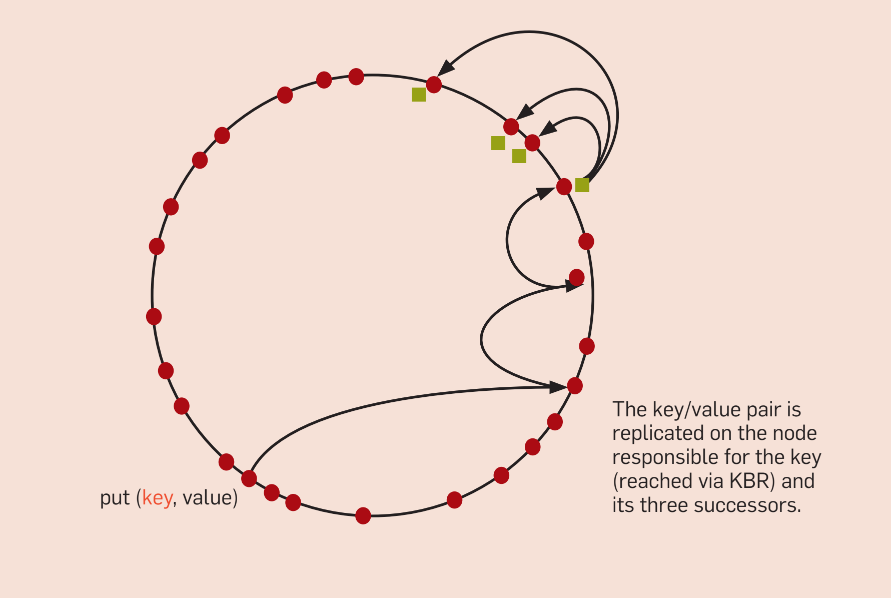
Often, the scope of the flood (that is, the maximal number of hops from the querying nodes that a flood message is forwarded) is limited to trade recall (the probability that an object that exists in the system is found) for overhead (the number of messages required by the flood). An alternative to flooding is for the querying node to send a request message along a random walk through the overlay.

Gnutella was the first example of a decentralized, unstructured network that used flooding to locate content in a file sharing system.

Structured overlays. In structured overlays, distributed state is maintained using a *distributed hash table* (DHT) abstraction. The DHT has the same put/get interface as a conventional hash table. Inserted key/value pairs are distributed among the participating nodes in the structured overlay using a simple placement function. For instance, that function can position replicas of the key/value pair on the set of *r* nodes whose identifiers succeed the key in the circular key space. Note that in our terminology, the values correspond to the state objects maintained by the system.

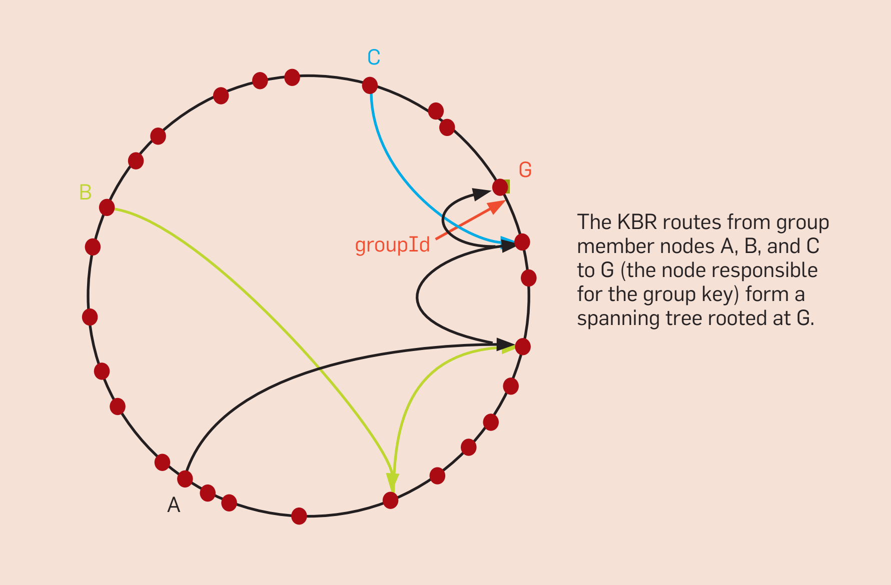
Given this replica placement policy, the DHT's put and get operations can be implemented using the KBR primitive in a straightforward manner. To insert (put) a key/value pair, we use the KBR primitive to determine the responsible node for the key *k* and store the pair on that node, which then propagates it to the set of replicas for *k*. To look up (get) a value, we use the KBR primitive to fetch the value associated with a given key. The responsible node can respond to the fetch request or forward it to one of the nodes in the replica set. Figure 3 shows an example put operation, where the value is initially pushed to the node responsible for key *k*, which is discovered using KBR, and this node pushes the value to its three immediate successors.

Figure 3. Inserting a value into a DHT.



The key/value pair is replicated on the node responsible for the key (reached via KBR) and its three successors.

Figure 4. An example KBR tree.



The KBR routes from group member nodes A, B, and C to G (the node responsible for the group key) form a spanning tree rooted at G.

When a DHT experiences churn, pairs have to be moved between nodes as the mapping of keys to nodes changes. To minimize the required network communication, large data values are typically not inserted directly into a DHT; instead, an indirection pointer is inserted under the value's key, which points to the node that actually stores the value.

DHTs are used, for instance, in file sharing networks such as eDonkey, and also in some versions of BitTorrent.

Summary. Unstructured overlays tend to be very efficient at locating widely replicated objects, while KBR-based techniques can reliably and efficiently locate any object that exists in the system, no matter how rare it may be. Put another way, unstructured overlays are good at finding "hay" while structured overlays are good at finding "needles." On the other hand, unstructured networks support arbitrary keyword-based queries, while KBR-based systems directly support only key-based queries.

Distributed coordination. Frequently, a group of nodes in a P2P application must coordinate their actions without centralized control. For instance, the set of nodes that replicate a particular object must inform each other of updates to the object. In another example, a node that is interested in receiving a particular streaming content channel may wish to find, among the nodes that currently receive that channel, one that is nearby and has available upstream network bandwidth. We will look at two distinct approaches to this problem: epidemic techniques where information spreads virally through the system, and tree-based techniques where distribution trees are formed to spread the information.

We focus only on decentralized overlays, since coordination can be accomplished by the controller node in partly centralized systems.

Unstructured overlays. In unstructured overlays, coordination typically relies on epidemic techniques. In these protocols, information is spread through the overlay in a manner similar to the way an infection spreads in a population: the node that produced the information sends it to (some of) its overlay neighbors, who send it to (some of) their neighbors, and so on. This method of dissemination is very simple and robust. As in all epidemic techniques, there is a trade-off between the speed of information dissemination and overhead. Moreover, if a given piece of information is of interest only to a subset of nodes and these nodes are widely dispersed within the overlay, then the information ends up being needlessly delivered to all nodes.

A more efficient way to coordinate the actions among a group of nodes is to form a spanning tree among the nodes. The spanning tree is embedded in the overlay graph, using a decentralized algorithm for spanning tree formation. This tree can then be used to multicast messages to all members, or to compute summaries (for example, sums, averages, minima, or maxima) of state variables within the group. However, this added coordination efficiency must be balanced against the overhead of maintaining the spanning tree in the unstructured overlay network.

Structured overlays. In structured overlays, spanning trees among any group of overlay nodes can be formed and maintained very efficiently using the KBR primitive, making trees the preferred method of coordination in these overlays. To join a spanning tree, a node uses KBR to route to a unique key associated with the group. The resulting union of the paths from all group members form a spanning tree rooted at the node responsible for the group's key. This *KBR tree* is then used to aggregate and disseminate state associated with the group, and to implement multicast and anycast. Figure 4 illustrates an example KBR tree formed by the union of the KBR routes from nodes *A*, *B*, and *C* to the key corresponding to the group id. This tree is rooted at node *G*, which is the responsible node for that key.

Because a join message terminates as soon as it intercepts the tree, group membership maintenance is decentralized, that is, the arrival or departure of a node is noted only by the node's parent and children in the tree. As a result, the technique scales to large numbers of groups, as well as large and highly dynamic groups.

Unstructured overlays are good at finding "hay," while structured overlays are good at finding "needles."

Summary. The epidemic techniques typically used for coordination in unstructured overlays are simple and robust to overlay churn, but they may not scale to large overlays or large numbers of groups, and information tends to propagate slowly. Spanning trees can increase the efficiency of coordination, but maintaining a spanning tree in an unstructured overlay adds costs.

The additional overhead for maintaining a *structured* overlay is proportional to the churn in the total overlay membership. Once that overhead is paid, KBR trees enable efficient and fast coordination among potentially numerous, large and dynamic subgroups within the overlay.

Content Distribution

Another common task in P2P systems is the distribution of bulk data or streaming content to a set of interested nodes. P2P techniques for content distribution can be categorized as tree-based (where fixed distribution trees are formed either with the aid of a structured overlay or embedded in an unstructured overlay), or swarming protocols (which have no notion of a fixed tree for routing content and usually form an unstructured overlay). Due to space constraints, we focus on the swarming protocols popularized by the BitTorrent protocol.[10]

In swarming protocols, the content is divided into a sequence of blocks, and each block is individually multicast to all overlay nodes such that different blocks are disseminated along different paths.

The basic operation of a swarming protocol is simple: once every swarming interval (say, one second), overlay neighbors exchange information indicating which content blocks they have available. (In streaming content distribution, only the most recently published blocks are normally of interest.) Each node intersects the availability information received from its neighbors, and then requests a block it does not already have from one of the neighbors who has it.

It is important that blocks are well distributed among the peers, to ensure neighboring peers tend to have blocks they can swap and that blocks remain available when some peers leave the system. To achieve such a distribution, the system can randomize both the choice of block to download and the choice of a neighbor from whom to request the block. In one possible strategy, a node chooses to download the rarest block among all blocks held by its overlay neighbors.[10]

The best known and original swarming protocol for bulk content distribution is BitTorrent.[10] Examples of swarming protocols used for streaming content include PPLive[17] and the original version of CoolStreaming.[38]

Challenges

Much of the promise of P2P systems stems from their independence of dedicated infrastructure and centralized control. However, these very properties also expose P2P systems to some unique challenges not faced by other types of distributed systems. Moreover, given the popularity of P2P systems, they become natural targets for misuse or attack. Here, we give an overview of challenges and attacks that P2P systems may face, and corresponding defense techniques. As you will see, some of the issues have been addressed to varying degrees, and others remain open questions.

Controlling membership. Most P2P systems have open or loosely controlled membership. This lack of strong user identities allows an attacker to populate a P2P system with nodes under his control, by creating many distinct identities (such action was termed a *Sybil attack*[15]). Once he controls a large number of "virtual" peers, an attacker can defeat many kinds of defenses against node failure or misbehavior, for example, those that rely on replication or voting. For instance, an attacker who wishes to suppress the value associated with some key *k* from a DHT can add virtual nodes to the system until he controls all of the nodes that store replicas of the value. These nodes can then deny the existence of that key/value pair when a get operation for key *k* is issued.

Initial proposals to address Sybil attacks required proof of work (for example, solving a cryptographic puzzle or downloading a large file) before a new node could join the overlay.[15,34] While these approaches limit the rate at which an attacker can obtain identities, they also make it more difficult for legitimate users to join. Moreover, an attacker with enough resources or access to a botnet can still mount Sybil attacks.

Much of the promise of P2P systems stems from their independence of dedicated infrastructure and centralized control. However, these very properties also expose P2P systems to some unique challenges not faced by other types of distributed systems.

Another solution requires certified identities,[7] where a trusted authority vouches for the correspondence between a peer identity and the corresponding real-world entity. The disadvantage of certified identities is that a trusted authority and the necessary registration process may be impractical or inappropriate in some applications.

Protecting data. Another aspect of P2P system robustness is the availability, durability, integrity, and authenticity of the data stored in the system or downloaded by a peer. Different types of P2P systems have devised different mechanisms to address these problems.

Integrity and authenticity. In the case of DHTs, data integrity is commonly verified using *self-certifying* named objects. DHTs take advantage of the fact that they have flexibility in the choice of the keys for values stored in the DHT. By setting *key=hash(value)* during the put operation, the downloader can verify the retrieved data is correct by applying the cryptographic hash function to the result of the get operation and comparing it to the original key. Systems that store mutable data and systems that allow users to choose arbitrary names for inserted content can instead use cryptographic signatures to protect the integrity and authenticity of the data. However, such systems require an infrastructure to manage the cryptographic keys.

Studies show that systems that do not protect the integrity of inserted data (including many file sharing systems) tend to be rife with mislabeled or corrupted content.[8,22] One possible approach to counter the problem of content pollution is for peers to vote on the authenticity of data. For example, a voting system called Credence was developed by researchers and used by several thousands of peers in the Gnutella file sharing network.[34] However, the problem remains challenging given the possibility of Sybil attacks to defeat the voting.

Availability and durability. The next challenge is how to ensure the availability and durability of data stored in a P2P system. Even in the absence of attacks, ensuring availability can prove difficult due to churn. For a data object to be available, at least one node that stores a replica must be online at all times. To make sure an object remains available under churn, a system must constantly move replicas to live nodes, which can require significant network bandwidth. For this reason, a practical P2P storage system cannot simultaneously achieve all three goals of scalable storage, high availability, and resilience to churn.[6]

Another challenge is that the long-term membership of a P2P storage system (that is, the set of nodes that periodically come online) must be non-decreasing to ensure the durability of stored data. Otherwise, the system may lose data permanently, since the storage space available among the remaining members may fall below that required to store all the data.

Incentives. Participants in a P2P system are expected to contribute resources for the common good of all peers. However, users don't necessarily have an incentive to contribute if they can access the service for free. Such users, called free riders, may wish to save their own disk space, bandwidth, and compute cycles, or they may prefer not to contribute any content in a file-sharing system.

Free riding is reportedly widespread in many P2P systems. For instance, in 2000 and 2001, studies of the Gnutella system found a large fraction of free riders.[2,28] More recently, a study of a DHT used in the eMule file-sharing system found large clusters of peers (with more than 10,000 nodes) that had modified their client software to produce the same node identifier for all nodes, which means these nodes are not responsible for any keys.[31]

The presence of many free riders reduces the resources available to a P2P system, and can deteriorate the quality of the service the system is able to provide to its users. To address this issue, incentive schemes have been incorporated in the design of P2P systems.

BitTorrent uses a tit-for-tat strategy, where to be able to download a file from a peer, a peer must upload another part of the same file in return, or risk being disconnected from that peer.[10] This provides a strong incentive for users to share their upload bandwidth, since a peer that does not upload data will have poor download performance. A number of other incentive mechanisms have been proposed, which all try to tie the quality of the service a peer receives to how much that peer contributes.[12,25]

Managing P2P systems. Whether P2P systems are easier to manage than other distributed systems is an open question.

On the one hand, P2P systems adapt to a wide range of conditions with respect to workload and resource availability, they automatically recover from most node failures, and participating users look after their hardware independently. As a result, the burden associated with the day-to-day operation of P2P systems appears to be low compared to server-based solutions, as evidenced by the fact that graduate students have been able to deploy and manage P2P systems that attract millions of users.[16]

On the other hand, there is evidence that P2P systems can experience widespread disruptions that are difficult to manage. For instance, on Aug. 16, 2007, the Skype overlay network collapsed and remained unavailable for several days. The problem was reportedly triggered by a Microsoft Windows Update patch that caused many of the peers to reboot around the same time, causing a lack of resources that, combined with a software bug, prevented the overlay from recovering.[30] This type of problem may indicate the lack of centralized control over available resources and participating nodes makes it difficult to manage systemwide disruptions when they occur. However, more research and long-term practical experience with deployed systems is needed to settle this question.

Some of the challenges P2P systems face (for example, data integrity and authenticity) are largely solved, while others (for example, membership control and incentives) have partial solutions that are sufficient for important applications. However, some problems remain wide open (for example, data durability and management issues). Progress on these problems may be necessary to further expand the range of applications of P2P technology.

Peer-to-Peer and ISPs

Internet service providers have witnessed the success of P2P applications with mixed feelings. On one hand, P2P is fueling demand for network bandwidth. Indeed, P2P accounts for the majority of bytes transferred on the Internet.[29] On the other hand, P2P traffic patterns are challenging certain assumptions that ISPs have made when engineering their networks and when pricing their services.

To understand this tension, we must consider the Internet's structure and pricing. The Internet is a roughly hierarchical conglomeration of independent network providers. Local ISPs typically connect to regional ISPs, who in turn connect to (inter-)national backbone providers. ISPs at the same level of the hierarchy (so-called peer ISPs) may also exchange traffic directly. In particular, the backbone providers are fully interconnected.

Typically, peer ISPs do not charge each other for traffic they exchange directly, but customers pay for the bits they send to their providers. An exception is residential Internet connections that are usually offered at a flat rate by ISPs.

This pricing model originated at a time when client-server applications dominated the traffic in the Internet. Commercial server operators pay their ISPs for the bandwidth used, who in turn pay their respective providers. Since residential customers rarely operate servers (in fact, their terms of use do not allow them to operate commercial servers), it was reasonable to assume they generate little upstream traffic, keeping costs low for local ISPs and enabling them to offer flat-rate pricing.

With P2P content distribution applications, however, residential P2P nodes upload content to each other. Unless the P2P nodes happen to connect to the same ISP or to two ISPs that peer directly with each other, the uploading node's ISP must forward the data to its own provider. This incurs costs that the ISP cannot pass on to its flat-rate customers.[20] As a result of this tension, some ISPs have started to traffic shape and even block BitTorrent traffic.[14] Whether network operators should be required to disclose such practices, and if they should be allowed at all to discriminate among different traffic types is the subject of an ongoing debate.

Independent of the outcome of this debate, the tension will have to be resolved in a way that allows P2P applications to thrive while ensuring the profitability of ISPs. A promising technical approach is to bias the peer selection in P2P applications toward nodes connected to the same ISP or to ISPs that peer with each other.[20] Another solution is for ISPs to change their pricing model.

A more fundamental tension is that some ISPs view many of the currently deployed P2P applications as competing with their own value-added services. For instance, ISPs that offer conventional telephone service may view P2P VoIP applications as competition, and cable ISP may view P2P IPTV applications as competing with their own IPTV services. In either case, such ISP's market share in the more profitable value-added services is potentially diminished in favor of carrying more plain bits.

In the long term, however, ISPs will likely benefit, directly and indirectly, from the innovation and emergence of new services that P2P systems enable. Moreover, ISPs may find new revenue sources by offering infrastructure support for successful services that initially developed as P2P applications.

Conclusion

In this article, we have sketched the promise, technology, and challenges of P2P systems. As a disruptive technology, P2P creates significant opportunities and challenges for the Internet, industry, and society. Arguably the most significant promise of P2P technology lies in its ability to significantly lower the barrier for innovation. But the great strength of P2P, its independence of dedicated infrastructure and centralized control, may also be its weakness, as it creates new challenges that must be dealt with through technical, commercial, and legal means.

One possible outcome is that P2P will turn out to be especially valuable as a proving ground for new ideas and services, in addition to keeping its role as a platform for grassroots services that enable free speech and the unregulated exchange of information. Services that turn out to be popular, legal, and commercially viable may then be transformed into more infrastructure-based, commercial services. Here, ideas from P2P systems may be combined with traditional, centralized approaches to build highly scalable and dependable systems.

References

1. About Skype: 100 Billion Skype-to-Skype Minutes Served; http://about.skype.com/2008/02/100_billion_skypetoskype_minut.html.
2. Adar, E. and Huberman, B.A. Free riding on Gnutella. *First Monday 5*, 10 (Oct. 2000).
3. Anderson, D.P. BOINC: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing* (2004), 4–10.
4. BBC News. One million viewers use iPlayer. http: //news.bbc.co.uk/2/hi/technology/7187967.stm.
5. Bittorrent (protocol). Wikipedia; http://en.wikipedia.org/wiki/BitTorrent_(protocol)#Adoption.
6. Blake, C. and Rodrigues, R. High availability, scalable storage, dynamic peer networks: Pick two. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems* (May 2003).
7. Castro, M., Druschel, P., Ganesh, A., Rowstron, A. and Wallach, D.S. Security for structured peer-to-peer overlay networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Dec. 2002).
8. Christin, N., Weigend, A.S. and Chuang, J. Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *Proceedings of the 6th ACM Conference on Electronic Commerce* (June 2005).
9. Clarke, I., Sandberg, O., Wiley, B. and Hong, T.W. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the Designing Privacy Enhancing Technologies—International Workshop on Design Issues in Anonymity and Unobservability* (July 2000).
10. Cohen, B. Incentives build robustness in BitTorrent. In *Proceedings of the 1st International Workshop on Economics of P2P Systems* (June 2003).
11. Cox, L.P. Murray, C.D. and Noble, B.D. Pastiche: Making backup cheap and easy. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Dec. 2002).
12. Cox, L.P. and Noble, B.D. Samsara: honor among thieves in peer-to-peer storage. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles* (Oct. 2003).
13. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W. Dynamo: Amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles* (Oct. 2007).
14. Dischinger, M., Mislove, A. Haeberlen, A. and Gummadi, K.P. Detecting BitTorrent blocking. In *Proceedings of the 8th Internet Measurement Conference* (Oct. 2008).
15. Douceur, J. The Sybil attack. In *Proceedings of the First International Workshop on Peer-to-Peer Systems* (Mar. 2002).
16. Freedman, M.J., Freudenthal, E. and Mazières, D. Democratizing content publication with Coral. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation* (Mar. 2004).
17. Hei, X., Liang, C., Liang, J., Liu, Y. and Ross, K.W. Insights into PPLive: A measurement study of a large-scale P2P IPTV system. In *Proceedings of the 15th International World Wide Web Conference, IPTV Workshop* (May 2006).
18. Hildrum, K., Kubiatowicz, J.D., Rao, S. and Zhao, B.Y. Distributed object location in a dynamic network. In *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures* (2002), 41–52.
19. Huebsch, R., Hellerstein, J.M., Lanham, N., Loo, B.T, Shenker, S. and Stoica, I. Querying the Internet with PIER. In *Proceedings of the 29th International Conference on Very Large Data Bases* (Sept. 2003).
20. Karagiannis, T., Rodriguez, P., and Papagiannaki, K. Should Internet service providers fear peer-assisted content distribution? In *Proceedings of the Internet Measurement Conference* (Oct. 2005).
21. Li, B., Xie, S., Qu, Y., Keung, G., Lin, C., Liu, J. and Zhang, X. Inside the new coolstreaming: Principles, measurements and performance implications. In *Proceedings of INFOCOM* (2008).
22. Liang, J., Kumar, R., Xi, Y. and Ross, K.W. Pollution in P2P file sharing systems. In *Proceedings of INFOCOM* (Mar. 2005).
23. Maniatis, P., Roussopoulos, M., Giuli, T.J., Rosenthal, D.S.H. and Baker, M. The LOCKSS peer-to-peer digital preservation system. *ACM Transactions on Computer Systems 23*, 1 (2005), 2–50.
24. Mislove, A. Post, A. Haeberlen, A. and Druschel, P. Experiences in building and operating ePOST, a reliable peer-to-peer application. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems* (Apr. 2006).
25. Nandi, A., Ngan, T-W.J, Singh, A., Druschel, P. and Wallach, D.S. Scrivener: Providing incentives in cooperative content distribution systems. In *Proceedings of the ACM/IFIP/USENIX 6th International Middleware Conference* (Nov. 2005).
26. Renesse, R.V, Birman, K.P. and Vogels, W. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems 21*, 2 (2003), 164–206.
27. Rowstron, A. and Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms* (Nov. 2001).
28. Saroiu, S., Gummadi, P.K., and Gribble, S.D. A measurement study of peer-to-peer file sharing systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking* (Jan. 2002).
29. Skinkers: Enterprise communication management; http://www.skinkers.com/About_us/About_Skinkers.
30. Skype: What happened on August 16; http://heartbeat.skype.com/2007/08/what_happened_on_august_16.html.
31. Steiner, M., Biersack, E.W. and Ennajjary, T. Actively monitoring peers in KAD. In *Proceedings of the 6th International Workshop on Peer-to-Peer Systems* (Feb. 2007).
32. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of SIGCOMM '01*, (Aug. 2001).
33. Stribling, J. Li J., Councill, I.G., Kaashoek, M.F. and Morris, R. Overcite: A distributed, cooperative citeseer. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation* (May 2006).
34. Walsh, K. and Sirer, E.G. Experience with an object reputation system for peer-to-peer filesharing. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation* (May 2006).
35. Wang, L., Park, K., Pang, R., Pai, V.S., and Peterson, L. Reliability and security in the CoDeeN content distribution network. In *Proceedings of the USENIX 2004 Annual Technical Conference* (June 2004).
36. Wilcox-O'Hearn, B. Experiences deploying a large-scale emergent network. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems* (Mar. 2002).
37. Yalagandula, P. and Dahlin, M. A scalable distributed information management system. In *Proceedings of SIGCOMM '04* (2004).
38. Zhang, X., Liu, J., Li, B. and Yum, T-S.P. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proceedings of INFOCOM '05* (2005).

Rodrigo Rodrigues (rodrigo@mpi-sws.org) is a tenure-track faculty member at the Max Planck Institute for Software Systems (MPI-SWS), where he heads the dependable systems group.

Peter Druschel (druschel@mpi-sws.org) is the founding director of the Max Planck Institute for Software Systems (MPI-SWS), where he heads the distributed systems group.